Integration von Funktionen in das Konfigurationstool für Automatik-Funktionen (ATEO)

Version vom 17. September 2011

Die aktuelle Version dieses Dokuments ist im GIT-Repository des ATEO-Projekts zu finden:
http://www.assembla.com/code/ATEO/git/nodes/doc

Inhaltsverzeichnis

1 Einführung

In dieser Anleitung werden die einzelnen Schritte beschrieben, die durchgeführt werden müssen, um einen neuen AAF-Agenten im Automatiken-GUI als Funktion zur Verfügung zu stellen.

Abbildung 1 zeigt das Automatiken-GUI mit den für diese Anleitung zwei wichtigen Bereichen. Links ist der Funktionsbereich, in dem elementare und kombinierte Funktionen angezeigt werden. Rechts ist der Konfigurationsbereich einer Funktion.

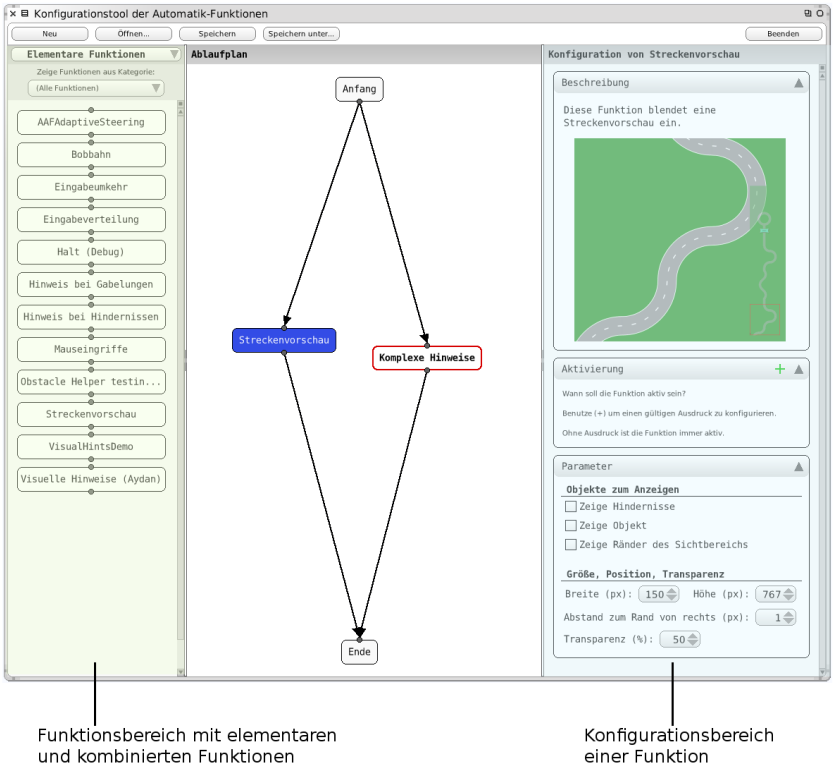

Abbildung 1: Das Automatiken-GUI mit hervorgehobenen Funktions- und Konfigurationsbereich.

2 Erstellen eines passenden Agenten

Um einen AAF-Agenten im Automatiken-GUI integrieren zu können, muss er zunächst erstellt werden. Hierbei sind einige Bedingungen zu erfüllen, damit der fertig implementierte Agent ins Automatiken-GUI eingebunden und später in einer Automatik verwendet werden kann.

Die Quelltext-Beispiele sind hauptsächlich aus den Klassen AAFMouseInputAgent und AAFMouseInputAgentDialog entnommen. Dabei wurde die Formatierung geringfügig verändert um die Darstellung in diesem Dokument zu optimieren.

2.1 Ableiten von AAFAgent

Der Agent muss eine Ableitung von AAFAgent sein. Dabei muss die Ableitung nicht direkt sein, es ist ebenso möglich, über eine oder mehrere Zwischenklassen von AAFAgent zu erben.

```
AAFAgent subclass: #AAFMouseInputAgent
    instanceVariableNames: 'mwi scaleFactorX scaleFactorY'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'AAF-Agents-Dev'
```

2.2 Klartextname des Agenten

Damit das Automatiken-GUI die Funktion mit einer passenden Aufschrift versehen kann, muss der Klartextname des Agenten von der Klassenmethode plaintextName als Zeichenkette zurückgeliefert werden.

```
plaintextName
    ^ 'Mauseingriffe'
```

2.3 Einsatzbereitschaft anzeigen

Ist der Agent fertig implementiert und prinzipiell als Teil einer Automatik einsatzfähig, so muss dies angezeigt werden. Für diesen Zweck haben AAF-Agenten eine Klassenmethode readyForUse, welche einen booleschen Wert zurückgibt. Die geerbte Standardimplementierung liefert den Wert false. Soll der Agent für die Verwendung freigegeben werden, so muss der Programmierer diese Methode überschreiben und den Wert true zurückgeben.

```
readyForUse
    ^ true
```

Für Details zu den Anforderungen, die ein AAF-Agent erfüllen muss, um als Teil eines AAF-Graphen einsetzbar zu sein, sei hier auf die Dokumentation des AAF verwiesen.

Sind die beiden bis hierher genannten Voraussetzungen erfüllt, so wird im Automatiken-GUI bereits eine Funktion für den Agenten erstellt und im Funktionsbereich bereitgestellt.

Besitzt der Agent keinerlei Eigenschaften, die über die geerbten hinausgehen, so ist die Integration in das Automatiken-GUI an dieser Stelle abgeschlossen und die erstellte Funktion kann verwendet werden. Besitzt er weitere Eigenschaften, so sind noch weitere Schritte nötig, damit diese gespeichert, geladen und angepasst werden können.

2.4 Kategorien des Agenten

In der Automatiken-GUI kann die Anzeige von elementaren Funktionen im Funktionsbereich durch vergebene Kategorien gefiltert werden. Einem Agenten können mehrere Kategorien zugeordnet werden. Diese werden als eine SequenceableCollection, zum Beispiel ein Array, in der Klassenmethode tags des Agenten festgelegt:

```
tags
    ^ #('(Entwicklung)')
```

2.5 Abfrage und Setzen spezieller Eigenschaften

Besitzt der Agent Eigenschaften, die über die geerbten hinausgehen, so muss er die Instanzmethoden getAllProps und setAllProps erweitern. Diese bilden die Schnittstelle zum Auslesen und Setzen aller Eigenschaften, ohne die einzelnen Zugriffsmethoden kennen zu müssen. Benötigt wird diese Schnittstelle z. B. zum Speichern und Wiederherstellen von Automatiken.

getAllProps muss ein Dictionary zurückgeben, welches die Eigenschaften und ihre aktuellen Werte als Schlüssel-Wert-Paare enthält. Hierbei ist zu beachten, dass gleich zu Beginn die entsprechende Methode der Basisklasse aufgerufen werden muss, sodass auch die Eigenschaften enthalten sind, die geerbt wurden.

```
getAllProps
    | dict |
    dict := super getAllProps.
    dict at: 'mwi' put: self mwi.

    ^ dict
```

setAllProps erhält ein Dictionary als Parameter. Dieses enthält alle Eigenschaften des Agenten sowie ihre zu setzenden Werte als Schlüssel-Wert-Paare. Die Methode setAllProps ist nun dafür zuständig, die Werte über die entsprechenden Zugriffsmethoden den Eigenschaften zuzuweisen. Auch hier ist zu beachten, dass ein Teil der Werte durch die setAllProps-Methode der Basisklasse verarbeitet werden muss, um die geerbten Eigenschaften zu setzen.

```
setAllProps: aPropertyDictionary
    (aPropertyDictionary size >= 2)
    ifTrue: [
        super setAllProps: aPropertyDictionary.
        self mwi: (AAFUtils convert:
            (aPropertyDictionary at: 'mwi') type: 'Number').
    ].
```

An dieser Stelle ist der Agent so weit, dass er tatsächlich im Automatiken-GUI verwendet werden kann, auch Speichern und Laden sind möglich. Allerdings kann er bis hierher nur mit seinen Standardeigenschaften verwendet werden, welche sinnvoll vorbelegt sein sollten, da es noch keine Möglichkeit zum Konfigurieren gibt.

3 Erstellen des Konfigurationsdialogs

Damit die Eigenschaften des Agenten angezeigt und konfiguriert werden können, ist es notwendig, für den Agenten einen passenden Dialog zu erstellen. Die hierfür durchzuführenden Schritte werden im Folgenden erläutert.

3.1 Ableiten von AAFAgentDialog

Der Konfigurationsdialog für den Agenten muss eine Ableitung von AAFAgentDialog sein. Auch hier spielt es wieder keine Rolle, ob die Ableitung direkt oder über Zwischenklassen erfolgt.

```
AAFAgentDialog subclass: #AAFMouseInputAgentDialog
    instanceVariableNames: 'dropDownMenu'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'AAFGT-Dialog'
```

3.2 Benennung der Dialogklasse

Damit die Dialogklasse vom Automatiken-GUI gefunden und verwendet werden kann, muss sie den gleichen Namen tragen, wie der zugehörige Agent, erweitert durch die Zeichenkette „Dialog“. Heißt also die Agenten-Klasse beispielsweise MouseInputAgent, so heißt die zugehörige Dialogklasse MouseInputAgentDialog.

3.3 Hinzufügen der Bedienelemente für die speziellen Eigenschaften

Um Eigenschaften anzeigen und editieren zu können, die der Agent über die geerbten hinaus besitzt, muss die Methode addSpecialElements entsprechend erweitert werden. In dieser Klasse werden die entsprechenden Bedienelemente erzeugt und anschließend dem dafür vorgesehenen Bereich buttonArea zugefügt. Zuletzt werden die angezeigten Werte aktualisiert, indem sie vom Agenten abgefragt werden.

```
addSpecialElements
    "Sets up parts which are unique for this type of dialog."
    | container |
    dropDownMenu := AAFGTDropDownMenu
        value: 'MWB1' items: #('MWB1' 'MWB2')
        target: self action: #userChose:.

    container := AlignmentMorph newRow color: Color transparent.
    container
        addMorphBack: (AAFGTOneLineLabel contents:
            'Maussteuerung aktivieren für: ');
        addMorphBack: dropDownMenu.

    self buttonArea
        addMorph: (AAFGTWidgetUtils centered: container).

    self updateFromDelegate.
```

Die Erstellung der Bedienelemente kann nicht allgemein beschrieben werden, da sie stark davon abhängig ist, um welchen Datentyp es sich handelt und welche Art von Bedienelement zum Einsatz kommen soll. Die Bedienelemente auf die zurückgegriffen

werden kann, finden sich in der *AAFGT Widget Gallery*. In einem Workspace kann diese via AAFGTWidgetGallery open. aufgerufen werden.

3.4 Aktualisieren der Anzeige bei Änderungen am Agenten

Damit der Dialog sich im Falle von Änderungen am Agenten korrekt aktualisiert, muss nun noch die Methode updateFromDelegate implementiert werden. Hierfür müssen die aktuellen Werte vom Agenten abgefragt und den entsprechenden Anzeigeelementen zugewiesen werden. Wichtig ist auch hier wieder der Aufruf der updateFromDelegate-Methode der Basisklasse, da nur so die Aktualisierung der Eigenschaften erfolgen kann, die der Agent geerbt hat.

```
updateFromDelegate
    | value |
    super updateFromDelegate.

    value := delegate mwi = 1
        ifTrue: [ 'MWB1' ]
        ifFalse: [ 'MWB2' ].

    dropDownMenu label: value.
```

Nun ist der Agent vollständig einsatzbereit und kann nicht nur gespeichert und geladen, sondern über den Dialog auch in seinen Eigenschaften angepasst werden.

3.5 Den Agenten dem Benutzer erklären

Durch die Implementierung von weiteren Methoden kann der Agent für den Benutzer besser erklärt werden.

Über die Klassenmethode shortDescription *der Agentenklasse* kann ein kleiner Hilfetext zurückgegeben werden:

```
shortDescription
    ^'Diese Funktion erlaubt die Steuerung eines',
     'Mikroweltbewohners mit der Maus'.
```

Verweilt der Benutzer mit der Maus über der Funktion im Funktionsbereich der Automatiken-GUI, dann erscheint dieser Hilfetext als Tooltip.

Über die Klassenmethode longDescription *der Dialogklasse* kann eine Hilfestellung zurückgegeben werden. Diese erscheint im Konfigurationsbereich der Funktion als eigener kleiner Abschnitt mit dem Titel "Beschreibung". Damit die Möglichkeit gegeben ist Bilder anzuzeigen, wird nicht nur ein einfacher String, wie bei shortDescription zurückgegeben, sondern ein Morph. Sei hierfür ausnahmsweise die Methode longDescription der Klasse AAFMapPreviewAgentDialog betrachtet:

```
longDescription
    | container |
    container := AlignmentMorph newColumn color: Color transparent.
    container cellInset: 10.
```

```
        container
            addMorphBack: (
                AAFGTMultiLineLabel contents:
                    'Diese Funktion blendet eine Streckenvorschau ein.'
            );
            addMorphBack: (
                (Form fromFileNamed: SAMConfig pathImagesHints,
                    'doc_mapPreview.png') asMorph
            ).

    ^ container
```

Der zurückgegebene Morph container in diesem Beispiel umschließt einen Text sowie ein Bild.